

PCT

WORLD INTELLECTUAL PROPERTY ORGANIZATION  
International Bureau



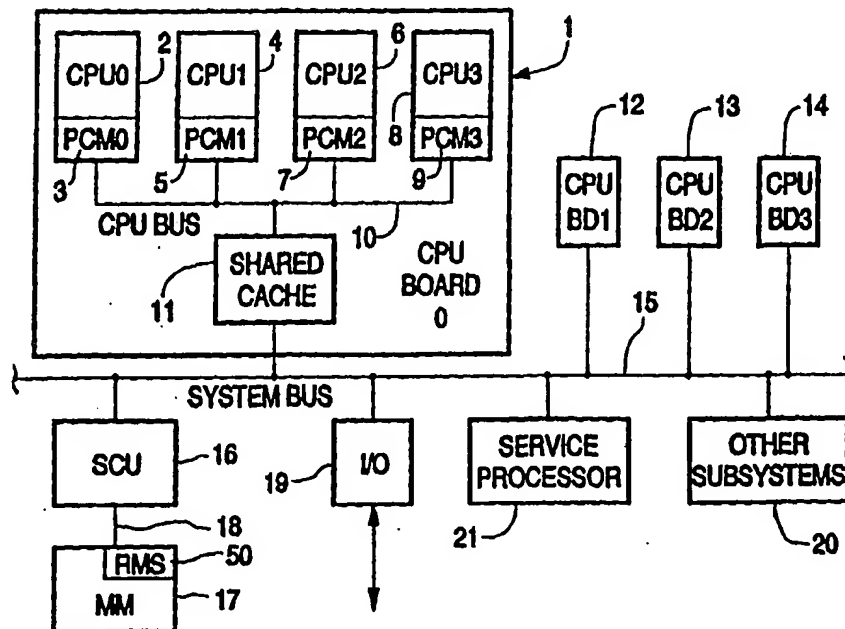
INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification <sup>6</sup> : <b>G06K</b>		(11) International Publication Number: <b>WO 98/25222</b>
<b>A2</b>		(43) International Publication Date: 11 June 1998 (11.06.98)
(21) International Application Number: PCT/US97/22185 (22) International Filing Date: 3 December 1997 (03.12.97) (30) Priority Data: 60/032,442 3 December 1996 (03.12.96) US (71) Applicant: BULL HN INFORMATION SYSTEMS INC. [US/US]; 300 Concord Road, Billerica, MA 01821 (US). (72) Inventors: ANDRESS, Sidney, L.; 5119 West Sweetwater, Glendale, AZ 85304 (US). MCCULLEY, Lowell, D.; 21653 North 58th Street, Glendale, AZ 85308 (US). (74) Agent: SOLAKIAN, John, S.; Bull HN Information Systems Inc., Law Office MA30-883A, 300 Concord Road, Billerica, MA 01821-4186 (US).		(81) Designated States: European patent (AT, BE, CH, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE).  Published <i>Without international search report and to be republished upon receipt of that report.</i>

(54) Title: FAULT INTERCEPT AND RESOLUTION PROCESS INDEPENDENT OF OPERATING SYSTEM

(57) Abstract

A fault handling process in a computer system subject to CPU design errors and functioning under an operating system (OS) having an integral fault handling module includes the steps of: setting an intercept flag when a central processor fault occurs if the fault is to be directed to a preprocessor; establishing a safestore frame which includes information identifying the type of fault and whether the intercept flag is set; and transferring control to the OS fault handling module; then in the OS fault handling module, determining whether the intercept flag is set; if the intercept flag is not set, handling the fault in the OS fault module; if the intercept flag is set, transferring control from the OS fault module to an Intercept Process written in machine language; and handling the fault in the Intercept Process. This renders the resolution of faults due to correctable CPU design errors independent of the OS employed at a given installation and customizable to a given system without the need to revise the OS fault modules for each OS. As each such design error is worked out (e.g., by installing a substitute integrated circuit in which the error has been corrected), the Intercept Process (and CPU firmware) can be modified to remove monitoring and handling for faults due to the corrected error.



**FOR THE PURPOSES OF INFORMATION ONLY**

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	ES	Spain	LS	Lesotho	SI	Slovenia
AM	Armenia	FI	Finland	LT	Lithuania	SK	Slovakia
AT	Austria	FR	France	LU	Luxembourg	SN	Senegal
AU	Australia	GA	Gabon	LV	Latvia	SZ	Swaziland
AZ	Azerbaijan	GB	United Kingdom	MC	Monaco	TD	Chad
BA	Bosnia and Herzegovina	GE	Georgia	MD	Republic of Moldova	TG	Togo
BB	Barbados	GH	Ghana	MG	Madagascar	TJ	Tajikistan
BE	Belgium	GN	Guinea	MK	The former Yugoslav Republic of Macedonia	TM	Turkmenistan
BF	Burkina Faso	GR	Greece	ML	Mali	TR	Turkey
BG	Bulgaria	HU	Hungary	MN	Mongolia	TT	Trinidad and Tobago
BJ	Benin	IE	Ireland	MR	Mauritania	UA	Ukraine
BR	Brazil	IL	Israel	MW	Malawi	UG	Uganda
BY	Belarus	IS	Iceland	MX	Mexico	US	United States of America
CA	Canada	IT	Italy	NE	Niger	UZ	Uzbekistan
CF	Central African Republic	JP	Japan	NL	Netherlands	VN	Viet Nam
CG	Congo	KE	Kenya	NO	Norway	YU	Yugoslavia
CH	Switzerland	KG	Kyrgyzstan	NZ	New Zealand	ZW	Zimbabwe
CI	Côte d'Ivoire	KP	Democratic People's Republic of Korea	PL	Poland		
CM	Cameroon	KR	Republic of Korea	PT	Portugal		
CN	China	KZ	Kazakhstan	RO	Romania		
CU	Cuba	LC	Saint Lucia	RU	Russian Federation		
CZ	Czech Republic	LI	Liechtenstein	SD	Sudan		
DE	Germany	LK	Sri Lanka	SE	Sweden		
DK	Denmark	LR	Liberia	SG	Singapore		
EE	Estonia						

# **FAULT INTERCEPT AND RESOLUTION PROCESS**

## **INDEPENDENT OF OPERATING SYSTEM**

### Cross Reference to Related Provisional Application

This application claims the benefit of the filing date of U.S. Provisional Patent Application Serial No. 60/032,442 filed December 3, 1996, entitled INTERCEPT PROCESS by Sidney L. Andress.

### Field of the Invention

This invention relates to computer central processors and, more particularly, to the repetitive temporary storage of central processing register contents and supporting information in a safestore in order to facilitate recovery from a fault or transfer to another domain. Still more particularly, this invention relates to a safestore feature which intercepts certain faults resulting from known system design errors and diverts the resolution process for handling such faults from the operating system fault handling facility to a special purpose software fault handling facility.

### Background of the Invention

As personal computers and workstations have become more and more powerful, makers of mainframe computers have undertaken to provide features which cannot readily be matched by these smaller machines in order to stay

1   viable in the marketplace. One such feature may be broadly referred to as fault  
2   tolerance which means the ability to withstand and promptly recover from  
3   hardware faults and other faults without the loss of crucial information. The  
4   central processing units (CPUs) of mainframe computers typically have error  
5   and fault detection circuitry, and sometimes error recovery circuitry, built in at  
6   numerous information transfer points in the logic to detect and characterize any  
7   fault which might occur.

8       The CPU(s) of a given mainframe computer comprises many registers  
9   logically interconnected to achieve the ability to execute the repertoire of  
10  instructions characteristic of the CPU(s). In this environment, the achievement  
11  of genuinely fault tolerant operation, in which recovery from a detected fault  
12  can be instituted at a point in a program immediately preceding the faulting  
13  instruction/operation, requires that one or more recent copies of all the  
14  software visible registers (and supporting information also subject to change)  
15  must be maintained and constantly updated. This procedure is typically carried  
16  out by reiteratively sending copies of the registers and supporting information  
17  (safestore information) to a special, dedicated memory or memory section.

18       When a fault occurs and analysis determines that recovery is possible,  
19  the safestore information is used to reestablish the software visible registers in

1 the CPU with the contents held recently before the fault occurred so that restart  
2 can be instituted or tried from the corresponding place in program execution.

3 The logical design of modern CPUs, particularly mainframes, is  
4 enormously complex. Inevitably, logic design errors are present as the design  
5 process proceeds. If the specific hardware in which a design error is  
6 discovered is still in development, it can simply be corrected, sometimes with  
7 appropriate changes in firmware. However, if the faulting condition occurs so  
8 rarely and is so elusive that it is only discovered after systems have been  
9 installed for commercial and/or other field operation, the correction of the  
10 hardware/firmware (for example, by replacing an integrated circuit having the  
11 design error with one in which the error has been corrected) can be time  
12 consuming. Similarly, if a rarely occurring hardware fault is discovered during  
13 development, there may be good reason, such as meeting delivery schedules, to  
14 forego any immediate attempt to effect a definitive hardware/firmware  
15 correction. In both instances, a conventional, and generally effective, prior art  
16 approach has been to set up the CPU firmware to detect and refer faults to a  
17 fault processing module written into the operating system.

18 There are, however, drawbacks to this approach. When design errors  
19 are discovered, the resolution process for the resulting fault must be

1 incorporated into the fault handling module operating system itself. This can  
2 be not only a formidable task, but the revisions to the operating system in all  
3 the systems in existence can be disruptive of normal operation. Further, some  
4 mainframe CPUs are configured to run under a plurality of operating systems.  
5 This requires changes to the fault processing modules of each operating system  
6 which can be accommodated by the CPUs. Still further, certain system design  
7 errors are often worked out, even after commercialization, as very large scale  
8 integrated circuits are modified and the chips changed out in individual  
9 installations. As a result, a feature in the operating system(s) introduced to  
10 handle a problem which no longer exists may adversely affect performance and  
11 certainly increases the amount of code in the operating system. It is to the  
12 solution of these related problems that the present invention is directed.

### 13 Objects of the Invention

14 It is therefore a broad object of this invention to provide, in a central  
15 processor, fault tolerant operation in which the storage and recovery of  
16 safestore information to handle certain faults is handled independent of the  
17 operating system.

18 It is a more specific object of this invention to provide a fault tolerant  
19 CPU in which the fault recovery process for certain predetermined faults is

1 diverted from the operating system fault processing module to an independent  
2 facility implemented in software written in machine specific language.

3 Summary of the Invention

4 Briefly, these and other objects of the invention are achieved, in a fault  
5 tolerant central processing unit having data manipulation circuitry including a  
6 plurality of software visible registers, by providing a safestore memory for  
7 storing the contents of the plurality of software visible registers, after a data  
8 manipulation operation, in order to facilitate restart after a detected fault by  
9 transferring the corresponding contents of the safestore memory back to the  
10 software visible registers during recovery from the detected fault. More  
11 particularly, the subject process is employed in a computer system functioning  
12 under an operating system (OS) having an integral fault handling module and  
13 includes the steps of: setting an intercept flag when a central processor fault  
14 occurs if the fault is to be directed to a preprocessor; establishing a safestore  
15 frame which includes information identifying the type of fault and whether the  
16 intercept flag is set; and transferring control to the OS fault handling module;  
17 then in the OS fault handling module, determining whether the intercept flag is  
18 set; if the intercept flag is not set, handling the fault in the OS fault module; if  
19 the intercept flag is set, transferring control from the OS fault module to an

1 Intercept Process written in machine language; and handling the fault in the  
2 Intercept Process.

3 This renders the resolution of faults due to correctable CPU design  
4 errors independent of the OS employed at a given installation and customizable  
5 to a given system without the need to revise the OS fault modules for each OS.  
6 As each such design error is worked out (e.g., by installing a substitute  
7 integrated circuit in which the error has been corrected), the Intercept Process  
8 (and CPU firmware) can be modified to remove monitoring and handling for  
9 faults due to the corrected error.

#### 10 Description of the Drawing

11 The subject matter of the invention is particularly pointed out and  
12 distinctly claimed in the concluding portion of the specification. The invention,  
13 however, both as to organization and method of operation, may best be  
14 understood by reference to the following description taken in conjunction with  
15 the subjoined claims and the accompanying drawing of which:

16 FIG. 1 is a high level block diagram of an multiprocessor computer  
17 system which is an exemplary environment for practicing the invention;

18 FIG. 2 is a slightly lower level block diagram showing additional details  
19 of an exemplary CPU board in the multiprocessor system of FIG. 1;



1        FIG. 3 is a block diagram showing additional details of a basic  
2        processing unit including within each CPU on the CPU board shown in FIG. 2;

3        FIG. 4 is a revised block diagram of the basic processing unit  
4        particularly showing the relationship of an auxiliary random access memory to  
5        the basic processing unit and its software visible registers (and supporting  
6        information) in accordance with the subject invention, the random access  
7        memory storing, inter alia, a Safestore Frame;

8        FIG. 5 is a system design flow chart of the intercept Process used in  
9        handling faults in accordance with the invention; and

10       FIG. 6 is a process flow diagram illustrating the Intercept Process and  
11       the cooperation between CPU hardware, CPU firmware, the operating system  
12       and the subject Intercept Process in handling faults.

13       Description of the Preferred Embodiment(s)

14       Attention is first directed to FIG. 1 which is a high level block diagram  
15       of an exemplary multiprocessor computer system incorporating the invention.  
16       A first CPU board (CPU Board "0") 1 includes four central processor units 2  
17       (CPU "0"), 4 (CPU "1"), 6 (CPU "2"), 8 (CPU "3"). Each of the central  
18       processor units 2, 4, 6, 8 situated on the first CPU board 1 includes an integral  
19       private cache memory module, 3, 5, 7, 9, respectively. The private cache

1 modules 3, 5, 7, 9 are each configured as "store into"; i.e., the results of each  
2 completed operation performed in the CPU are stored into the private cache.  
3 Each of the private cache modules 3, 5, 7, 9 on CPU Board "0" 1 interface  
4 with a CPU bus 10 for direct communications between the CPUs 2, 4, 6, 8.

5 In the exemplary system, there are three additional CPU boards 12  
6 (CPU Board "1"), 13 (CPU Board "2") and 14 (CPU Board "3"), each of  
7 which is substantially identical to CPU board 1 although those skilled in the  
8 multiprocessor art will understand that each CPU board and each CPU on each  
9 of the boards is assigned a unique identification number to facilitate  
10 communication and cooperation among the CPUs in the system.

11 CPU board 1 (i.e., CPU Board "0") also includes a shared cache 11  
12 disposed between ("bridging") the CPU bus 10 and a system bus 15. It will be  
13 understood that each of the CPU boards 12, 13, 14 also each includes a CPU  
14 bus and a shared cache, identically oriented.

15 A system control unit 16 serves to couple the system bus 15 to a main  
16 memory unit 17 via a memory bus 18. (It will be noted that the main memory  
17 unit 18 includes a Reserved Memory Space 50 - RMS - which will be  
18 discussed further below.) In addition, one or more input/output units 19  
19 interface the system bus 15 with various input/output subsystems, not shown,

1 to achieve input/output functions on a system basis, all as well known to those  
2 skilled in the art. Similarly, other subsystems 20, not otherwise specified or  
3 shown, may be connected to the system bus 15 to complete a given  
4 multiprocessor system. System control unit 16 also conventionally provides a  
5 multi-phase clock to all the system units requiring a common clock source. A  
6 service processor 21, typically a commercial personal computer or  
7 workstation, serves not only as a system and maintenance console, but also is  
8 used to boot the system and is employed extensively in analyzing and  
9 processing faults.

10 FIG. 2 is a slightly lower level block diagram of CPU "0" 2 of CPU  
11 board 1 (CPU Board "0") illustrating additional structure which is present in  
12 each CPU in the system. CPU "0" 2 includes a basic processing unit 22 and  
13 support circuitry 23 therefor.

14 As previously described, CPU "0" 2 also includes private cache module  
15 "0" 3 which constitutes a cache control unit 24 and a private cache 25 (which  
16 itself includes additional logic to be described below). Cache control unit 24  
17 includes paging unit 26, cache management unit 27 and CPU bus unit 28.  
18 Paging unit 26 interfaces with basic processing unit "0" 22 and cache  
19 management unit 27. Cache management unit 27 also interfaces with private

1 cache memory 25 and CPU bus unit 28. CPU bus unit also interfaces with  
2 CPU bus 10 and, via CPU bus 10, shared cache 11. Private cache 25 is also  
3 coupled directly to receive information from and send information to the CPU  
4 bus 10 and to receive information from and send information to basic  
5 processing unit "0" 22.

6 As previously described, shared cache 11 also interfaces with system  
7 bus 15 and, via system bus 15, with system control unit 16 and other  
8 systems/subsystems shown in FIG. 1. Main memory 17, including Reserve  
9 Memory Space 50, may be accessed via the system control unit 16 and  
10 memory bus 18.

11 It will be seen that there are numerous paths for information flow among  
12 the various blocks shown in FIGs. 1 and 2. The types of information may  
13 include control, address, instructions and operands. A given CPU may directly  
14 access its own private cache module and indirectly access the private cache  
15 modules incorporated into the other CPUs on a shared CPU board. Thus, CPU  
16 "0" 2 can access, via the CPU bus 10, the shared cache 11 it shares with CPU  
17 "1" 4, CPU "2" 6 and CPU "3" 8. CPU "0" 2 can also, under defined  
18 conditions, access the private cache module of CPU "2" 6 (for example) via  
19 the CPU bus 10 to effect a local "siphon". Further, CPU "0" 2 can access (via

1 CPU bus 10, shared cache 11 and system bus 15) the shared caches (not  
2 shown) on each of CPU Board "1" 12, CPU Board "2" 13 and CPU Board "3"  
3 14. Still further, a given CPU may indirectly access the private cache modules  
4 (not shown) of a CPU (not shown) on another CPU board; e.g., CPU "0" on  
5 CPU board 1 (CPU Board "0") may, under defined conditions, access the  
6 private cache module of any one of the CPUs on CPU Board "2" 13 (FIG. 1)  
7 via CPU bus 10, shared cache 11, system bus 15 and the shared cache on CPU  
8 Board "2" to effect a remote "siphon".

9 Further yet, for example, CPU "0" 2 can access main memory 17,  
10 including RMS 50, via CPU bus 10, shared cache 11, system bus 15, SCU 16  
11 and memory bus 18. Still further, for example, CPU "0" 2 can access, via  
12 CPU bus 10, shared cache 11 and system bus 15, any other block shown  
13 coupled to the system bus 15 in FIG. 1 to achieve bilateral communication with  
14 input/output devices, other subsystem components and even other  
15 multiprocessor systems.

16 FIG. 3 is a block diagram which includes additional details of a basic  
17 processing unit 22 in a system incorporating the present invention. The  
18 Address and Execution (AX) unit 30 is a microprocessing engine which  
19 performs all address preparation and executes all instructions except decimal

1 arithmetic, binary floating point and multiply/divide instructions. The main  
2 functions performed by the AX unit 30 include: effective and virtual address  
3 formation; memory access control; security checks; register change/use  
4 control; execution of basic instructions, shift instructions, security instructions,  
5 character manipulation and miscellaneous instructions; and CLIMB safestore  
6 file.

7       Efficient scientific calculation capability is implemented in the Floating  
8 Point (FP) coprocessor unit 34. The FP unit 34 executes all binary floating  
9 point arithmetic. This unit, operating in concert with the AX unit 30, performs  
10 scalar or vector scientific processing. The FP unit 34: executes all binary and  
11 fixed and floating point multiply and divide operations; computes 12 by 72-bit  
12 partial products in one machine cycle; computes eight quotient bits per divide  
13 cycle; performs modulo 15 residue integrity checks; executes all floating point  
14 mantissa arithmetic; executes all exponent operations in either binary or  
15 hexadecimal format; preprocesses operands and post-processes results for  
16 multiply and divide instructions; and provides indicator and status control.

17       The DN unit 32 performs the execution of decimal numeric Extended  
18 Instruction Set (EIS) instructions. It also executes Decimal-to-Binary (DTB),  
19 Binary-to-Decimal (BTD) conversion EIS instructions and Move-Numeric-Edit

1 (MVNE) EIS instructions in conjunction with the AX unit 30. The DN unit  
2 both receives operands from and sends results to the private cache 3. A  
3 COMTO ("command to") bus 38 and a COMFROM ("command from") bus  
4 36 couple together the AX unit 30, the DN unit 32 and the FP unit 34 for  
5 certain interrelated operations.

6 The AX unit 30 includes an auxiliary random access memory 40 which  
7 is used to store safestore (and other) information. Thus, the contents of the  
8 auxiliary RAM 40 are constantly updated with, for example, duplicates of the  
9 contents of software visible registers and other relevant information subject to  
10 change (collectively, the Safestore Frame or SSF) such that, in the event of the  
11 occurrence of a fault from which recovery has been determined to be possible,  
12 processing may be restarted at a point just prior to the fault by transferring the  
13 most recent register set stored in the auxiliary RAM 40 back to reestablish the  
14 register set.

15 The straightforward use of a safestore is known in the prior art as  
16 exemplified by U.S. Patent 5,276,862, entitled SAFESTORE FRAME  
17 IMPLEMENTATION IN A CENTRAL PROCESSOR by Lowell D.  
18 McCulley et al; U.S. Patent 5,553,232, entitled AUTOMATED SAFESTORE  
19 STACK GENERATION AND MOVE IN A FAULT TOLERANT CENTRAL

1   PROCESSOR by John E. Wilhite et al; and U.S. Patent 5,557,737 entitled  
2   AUTOMATED SAFESTORE STACK GENERATION AND RECOVERY  
3   IN A FAULT TOLERANT CENTRAL PROCESSOR by John E. Wilhite et  
4   al, all incorporated by reference herein for their disclosure of the repetitive  
5   storage of safestore information in a safestore memory and the use of safestore  
6   information in recovery from a fault.

7       As previously noted, the AX unit 30, DN unit 32 and FP unit 34 are,  
8   collectively, referred to as the basic processing unit (BPU) 22. Referring now  
9   to FIG. 4, it will be understood that the AX unit 30, (except for the auxiliary  
10   RAM 40), DN unit 32 and FP unit 34 and their support circuitry 23 (FIG. 2)  
11   are represented by the data manipulation logic block 42 in order that the  
12   auxiliary RAM 40 can be discussed in greater detail in the following discussion  
13   of the invention.

14       Intercept Process provides a fault preprocessor that can review fault  
15   situations and provide machine assembly language level assistance in managing  
16   system design problems. The provision of a fault handling module  
17   incorporated directly in a mainframe operating system to handle known design  
18   errors is well known and effectively permits full functionality operation of a  
19   system until a new hardware release is delivered, at which time the fault



1 handling module sections for handling the design errors which have been  
2 corrected can be disabled. However, there are conditions under which this  
3 basic approach has drawbacks. First, each fault correction routine in a fault  
4 handling module integral with an operating system remains until a new version  
5 of the operating system (a relatively rare event) even though the fault may have  
6 been corrected, for example, by substitution of an updated integrated circuit for  
7 one having the design fault. Second, the fault handling facility is not operating  
8 system independent for those machines which can run more than one operating  
9 system.

10 In accordance with the present invention, the preprocessor is invoked  
11 via CPU hardware/firmware to allow a common machine assembly language  
12 routine, stored in RMS 50, to function with a plurality of operating systems.  
13 (It may be noted that this technique is also useful in processing Service  
14 Processor related tasks.) A system design flow chart of the Intercept Process  
15 as adapted to the present invention is shown in FIG. 5.

16 Thus, as shown in FIG. 5, if a hardware design error is discovered, the  
17 CPU firmware is modified to set an Intercept flag (in the example, bit 7 of  
18 word 5 in the Safestore Frame) whenever the design error causes a fault. The  
19 Intercept Process code is built or modified to process the fault caused by the

1 design error. If and when the design error is later corrected and an integrated  
2 circuit chip having the design error has been replaced by a chip in which the  
3 error has been corrected, the CPU firmware is modified to eliminate checking  
4 for the design fault. If desired, the fault handling code for the fault can be  
5 removed from the Intercept Process code stored in RMS.

6 In accordance with the invention, the intercepting process is carried out  
7 using operating system software to detect hardware indicators requesting  
8 transfer to the special purpose Intercept Process machine assembly language  
9 code which is stored in RMS 50 by the Service Processor 21 during system  
10 initialization. As shown in FIG. 6, Intercept Process can, independent of the  
11 operating system in use, take corrective action then return to the faulting  
12 process or pass the fault back to the operating system fault module if no  
13 defined action for Intercept Process is detected. (It may be noted that the  
14 Intercept Process can also initiate certain tasks for the Service Processor, then  
15 return to the faulting process although this is not shown in FIG. 6 as this  
16 feature is not a part of the subject invention.)

17 Specific faults to be intercepted are established during the initialization  
18 of the CPU within its firmware. Thus, the Intercept flag can be set on any fault  
19 as determined to be necessary by the current processor firmware.

1       As previously described, the Intercept Process is a machine assembly  
2 language level process that is loaded into RMS 50 upon system initialization.  
3 By being resident in RMS and implemented in machine assembly language, the  
4 Intercept Process is functional with all operating systems which run on a given  
5 hardware system. The Intercept Process is configured such that any fault type  
6 can cause the CPU firmware to invoke it. This is done by setting a dedicated  
7 Intercept Fault flag in the "fault type" word of the SSF. Therefore, an  
8 Intercept Fault can "piggy-back" on a system level fault.

9       The Intercept Fault flag has priority over all system level faults. When  
10 the operating system detects the presence of the Intercept Fault flag, it transfers  
11 control to the Intercept Process before any processing of the current fault is  
12 performed. This will permit any corrective action of which the Intercept  
13 Process is capable to occur before an undesired recovery action is taken by the  
14 operating system's fault module.

15       The Intercept Process is customized for each release/version of the CPU  
16 firmware. This feature provides the ability for each release of processor  
17 firmware to specify the revision(s) to Intercept Process. This is a substantive  
18 improvement over the prior art in which the operating system fault module had  
19 to carry fixes for all known design errors which ever existed in the CPU design

1 (at least since the previous operating system release) because the operating  
2 system fault module could not determine the CPU firmware revision.

3 This process provides the ability to use CPU (under one or more  
4 proprietary operating systems) with known design errors and to quickly resolve  
5 customer problems while the hardware change is being developed. The  
6 Intercept Process is tied to the release of processor firmware. When the  
7 Service Processor loads the selected processor firmware during initialization, it  
8 also places the corresponding Intercept Process into Reserved Memory; thus,  
9 the firmware version and the Intercept Process version must match.

10 As previously noted, the Intercept Process resides in RMS 50 in main  
11 memory 17 starting on a page boundary and, in the example, is sixteen  
12 consecutive pages in size. These are real pages of memory defined by the  
13 Service Processor 21 when the Intercept Process is loaded during system  
14 initialization. The location of the Intercept Process in RMS is defined in a  
15 predetermined word of the system configuration area of RMS.

16 Preferably, the mapping of the real pages in RMS storing Intercept  
17 Process code is into the same working space as the operating system's fault  
18 module. Otherwise, the Intercept Process would be required to correct for  
19 problems in all working spaces in the system, and the changing of the working

1 space registers' contents would cause additional work for the operating system  
2 upon return from the Intercept Process.

3 The Intercept Process requires only limited use of the system registers to  
4 perform its function in order to correspondingly limit the impact upon the  
5 operating system when the Intercept Process is invoked. The following defines  
6 the descriptor and pointer registers used by the Intercept Process:

7 D0 = Return descriptor with pointer register having the location within  
8 the operating system's fault module.

9 D3 = Frames the SSF entry that has the Intercept Process flag set. The  
10 Intercept Process will look at data from the SSF to determine the  
11 corrective action required.

12 D4 = The Instruction Segment Register of the failed process. (There  
13 may be cases where the Intercept Process will have to examine the  
14 failing code as part of the recovery routine.)

15 In the example, the Intercept Process will transfer through "Pointer  
16 Register 0" to return to the operating system fault module. An "inter-segment  
17 transfer" is used to return to the operating system fault module to avoid  
18 affecting the Safestore Frame stack. As shown in FIG. 6, if there is no process  
19 for handling a given design fault resident in Intercept Process, a Transfer

1 through Pointer Register 0 by Intercept Process causes the operating system  
 2 fault module to try a restart of the faulting process (i.e., invoke a retry of the  
 3 failed instruction). (If it faults again, the problem may be referred to the  
 4 Service Processor, and a CPU freeze could result.) A Transfer through Pointer  
 5 Register 0-plus-one by Intercept Process causes the operating system fault  
 6 module to process the fault defined in the SSF entry.

7 Assuming that a patch for a given fault is resident in the current  
 8 Intercept Process version, the Intercept Process uses "Descriptor Register 3"  
 9 to access the faulting process SSF. Referring briefly to FIG. 4, to analyze the  
 10 reason for the request to the Intercept Process and to handle the fault, the  
 11 following data in the SSF is needed:

12	Firmware Address	(SSF Word 1)
13	IC of Fault	(SSF Word 3 or 4)
14	Fault Flags and Code	(SSF Word 5)
15	ISR type (ns/ei)	(SSF Word 8)
16	Index (X) Registers	(SSF Words 40-43)
17	A and Q Registers	(SSF Words 44-45)
18	Descriptors/Pointers	(SSF Words 48-53)

1       The A and Q Registers are the CPU's accumulator and supplementary  
2 accumulator registers, respectively. Other SSF information may be referenced.  
3       The Intercept Process, which modifies only limited information within the SSF,  
4 uses "Descriptor Register 4" to access the faulting process instruction stream.  
5       To understand the reason why the fault occurred requires an analysis of the  
6 instruction stream for conditions that can occur with the hardware signature  
7 value. The Intercept Process will not modify any instruction within the stream.

8       The Intercept Process, as necessary in the example, uses the A, Q, X1,  
9 X4 and X5 registers to perform its analysis of the intercepted fault. All  
10 program visible registers will be saved before they are used by the Intercept  
11 Process, and the contents of these registers will be restored to their original  
12 values before exiting the Intercept Process.

13       When any of the defined CPU faults occur, they proceed through the  
14 CPU fault priority logic and form a seven-bit fault code for the highest priority  
15 fault. After this step is performed in the normal way, the CPU fault firmware  
16 compares the seven-bit code to determine if the detected fault is in the group  
17 intended for "intercept" handling. If it is, bit 7 in SSF Word 5 is set "on"  
18 indicating that an intercept request is nested with this fault. The Intercept

1 Process is not introduced into hardware and therefore does not have any effect  
2 on existing fault priorities.

3 As shown in FIG. 6, when the Intercept Flag is set on in the current SSF  
4 Word 5, then the operating system will transfer to the first location of the  
5 Intercept Process code using a descriptor established during system  
6 initialization.

7 The code within the Intercept Process does not generate faults as the  
8 operating system fault module in the example is not able to handle a second  
9 fault (there is no hardware enforcement of this rule). In a multiprocessor  
10 system, the Intercept Process returns to the same CPU (conventionally  
11 identified by a System Identification Number or the equivalent) that initiated  
12 the request for service. This measure reduces the overhead of managing  
13 interrupts and faults within the Intercept Process.

14 Two special purpose instructions are used with the Intercept Process.  
15 These two instructions are used to provide certain processor mode permissions  
16 while executing within Intercept Process. The two special purpose instructions  
17 are SICPM (Set Intercept Mode) and RICPM (Reset Intercept Mode). The  
18 SICPM instruction is one of the first instructions executed by Intercept Process  
19 after control has been transferred to Intercept Process to ensure that the correct



1 CPU mode and permissions are set before any other instructions are executed.  
2 The RICPM instruction is one of the last instructions executed by the Intercept  
3 Process each time it is called to recover a fault. This instruction resets the  
4 processor mode and permissions to the state that they were in when the  
5 SICPM instruction was executed.

6 The Intercept Process enters at location zero of the first page that was  
7 defined to the operating system by the Service Processor when the system is  
8 initialized. This common entry point is the only entry point into the Intercept  
9 Process. The Intercept Process needs to identify itself to the system when it is  
10 in execution so that the hardware and firmware can automatically recover for  
11 errors encountered due to system mode and permissions. The Intercept  
12 Process identifies itself to the hardware by the execution of the previously  
13 discussed SICPM instruction. The execution of this instruction causes the  
14 hardware to set an internal flag indicating that the Intercept Process is  
15 temporarily in control of the CPU. A housekeeping routine saves the  
16 processor's program visible registers. In the example, the registers are saved  
17 by physical processor using X Register 7. This insures that the Intercept  
18 Process will have complete use of all program visible registers and still be able

1 to return to the operating system fault module with the registers in the state and  
2 with the content as when they were called.

3 The first step of processing an Intercept request is to determine the type  
4 of fault that caused the request. This is achieved by getting the fault type from  
5 the current SSF. DR3 is required to frame the current SSF. The seven-bit fault  
6 type generated by the CPU firmware is resident in bits 11 - 17 of Word 5 in the  
7 SSF. This value is used as an index into a processing table.

8 The next level of the Intercept Process execution routine is fetched from  
9 a functionality table. If the entry in this table for the fault type is not negative  
10 (bit zero = "0"), then the entry has the offset within the Intercept Process  
11 where the present fault type is to be processed. (If the entry in this table for  
12 the fault type is negative (bit zero = "1"), then there is no next level process  
13 routine, and control will be transferred to process an Service Processor I/O  
14 request. This feature is outside the present invention.)

15 In this manner, control is passed to the execution routine within  
16 Intercept Process for handling the specific fault type. It is at this level that the  
17 rules about the fault are applied to determine if the current fault qualifies for  
18 recovery. The rules, of course, vary depending upon the fault type and  
19 hardware problem(s) that caused the error conditions. When Intercept Process

1 support is required for a certain fault type, the fault handling function is  
2 appropriately defined.

3 The Intercept Process has two exits, and both cause the operating  
4 system to continue processing, but with different logic paths. A first exit from  
5 the Intercept Process causes the operating system to retry the failed instruction  
6 after the fault has been handled by Intercept Process. A second exit from the  
7 Intercept Process causes the operating system to continue its processing of the  
8 fault and to resolve it if possible. (Although not shown in FIG. 6, it is also  
9 possible that the operating system fault module will refer the fault to the  
10 Service Processor which might freeze the faulting CPU and reconfigure the  
11 system as necessary.)

12 Regardless of the exit point, the program visible registers must be  
13 restored before returning to the operating system's fault module. The Intercept  
14 flag (bit 7 in Word 5 in the SSF) will be reset as part of the wrap up processing  
15 in the Intercept Process.

16 As previously noted, the return from the Intercept Process to the  
17 operating system fault module is a lateral transfer using Pointer and Descriptor  
18 Register Zero. This type of transfer must be done since this is the same type of  
19 transfer used to enter the Intercept Process.

1       The foregoing description describes the invention in the environment of  
2   a multiprocessor computer system; however, it will be appreciated by those  
3   skilled in the art that the invention may be used with equal effect in a  
4   uniprocessor system which includes iterative execution instructions.

5       Thus, while the principles of the invention have now been made clear in  
6   an illustrative embodiment, there will be immediately obvious to those skilled  
7   in the art many modifications of structure, arrangements, proportions, the  
8   elements, materials, and components, used in the practice of the invention  
9   which are particularly adapted for specific environments and operating  
10   requirements without departing from those principles.

## WHAT IS CLAIMED IS:

1 1. In a computer system functioning under an operating system including a  
2 fault handling module, a process for handling a central processor fault  
3 comprising the steps of:

4 A) when a central processor fault occurs during an operation:

5 1) setting an intercept flag if the fault is to be directed to a  
6 preprocessor;

7 2) establishing a safestore frame which includes information  
8 identifying the type of fault and whether the intercept flag is set;  
9 and

10 3) transferring control to the operating system fault handling  
11 module;

12 B) in the operating system fault handling module, determining whether  
13 the intercept flag is set;

14 C) if the intercept flag is not set, handling the fault in the operating  
15 system fault module and going to step F);

16 D) if the intercept flag is set, transferring control from the operating  
17 system fault module to an intercept process written in machine language;

- 18           E) handling the fault in the intercept process; and  
19           F) retrying the operation which caused the fault in the central processor.

1   2. The process of Claim 1 in which modifiable central processor firmware is  
2   employed to sense the central processor fault and to selectively set the  
3   intercept flag while establishing the safestore frame.

1   3. The process of Claim 2 in which the central processor firmware is  
2   configured to recognize a central processor fault which is due to a known  
3   hardware design error.

1   4. The process of Claim 3 in which, after the known hardware design error has  
2   been corrected, the central processor firmware is reconfigured to eliminate  
3   monitoring for faults due to the known hardware design error.

1   5. The process of Claim 4 in which the current version of the central processor  
2   firmware and the current version of the intercept process are each matched to a  
3   central processor hardware release currently in use for a given central  
4   processor.

1 6. The process of Claim 2 in which the intercept process and central processor  
2 firmware are loaded upon initialization of the system in which the central  
3 processor is resident.

1 7. The process of Claim 3 in which the intercept process and central processor  
2 firmware are loaded upon initialization of the system in which the central  
3 processor is resident.

1 8. The process of Claim 4 in which the intercept process and central processor  
2 firmware are loaded upon initialization of the system in which the central  
3 processor is resident.

1 9. The process of Claim 5 in which the intercept process and central processor  
2 firmware are loaded upon initialization of the system in which the central  
3 processor is resident.

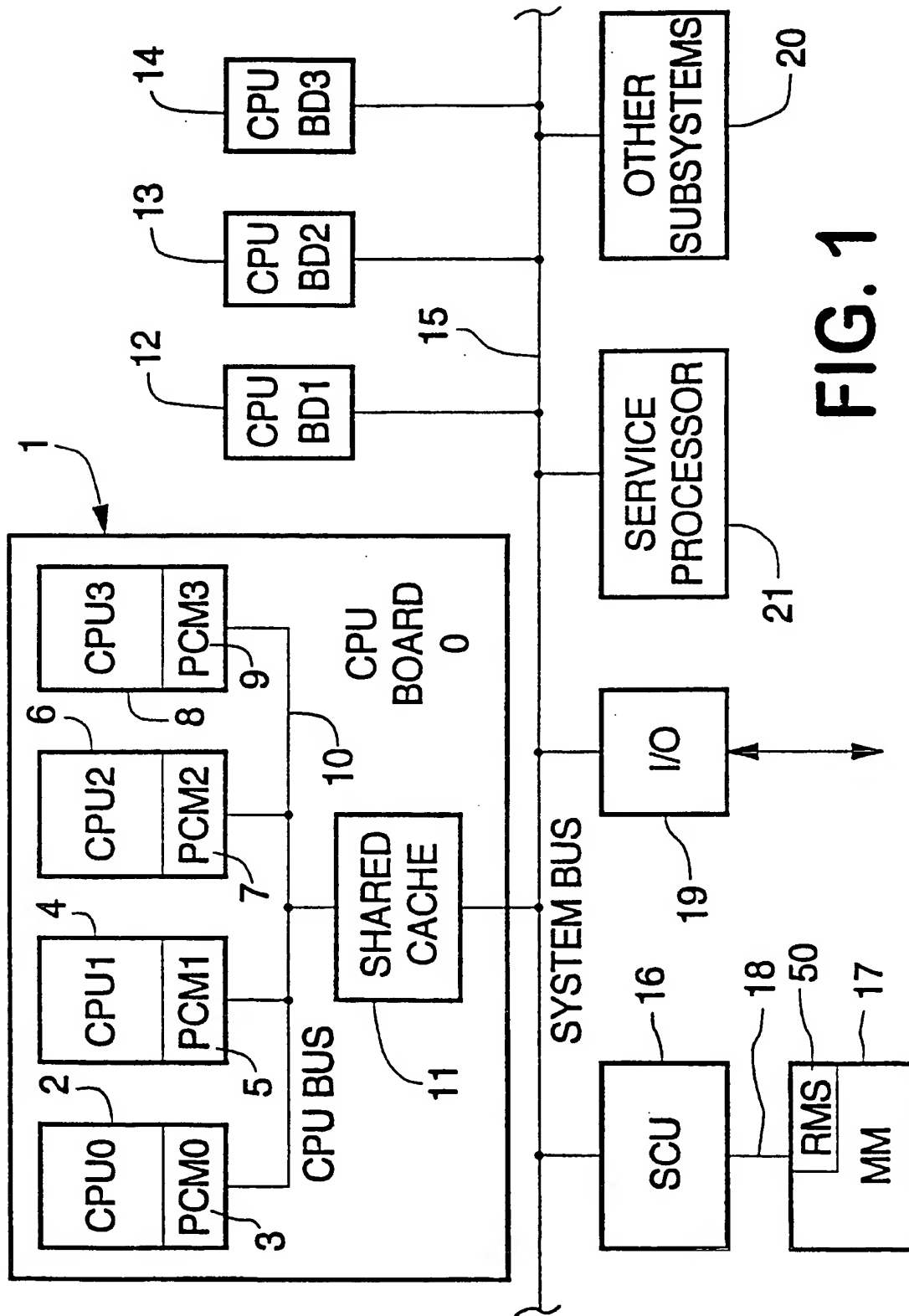


FIG. 1



2/5

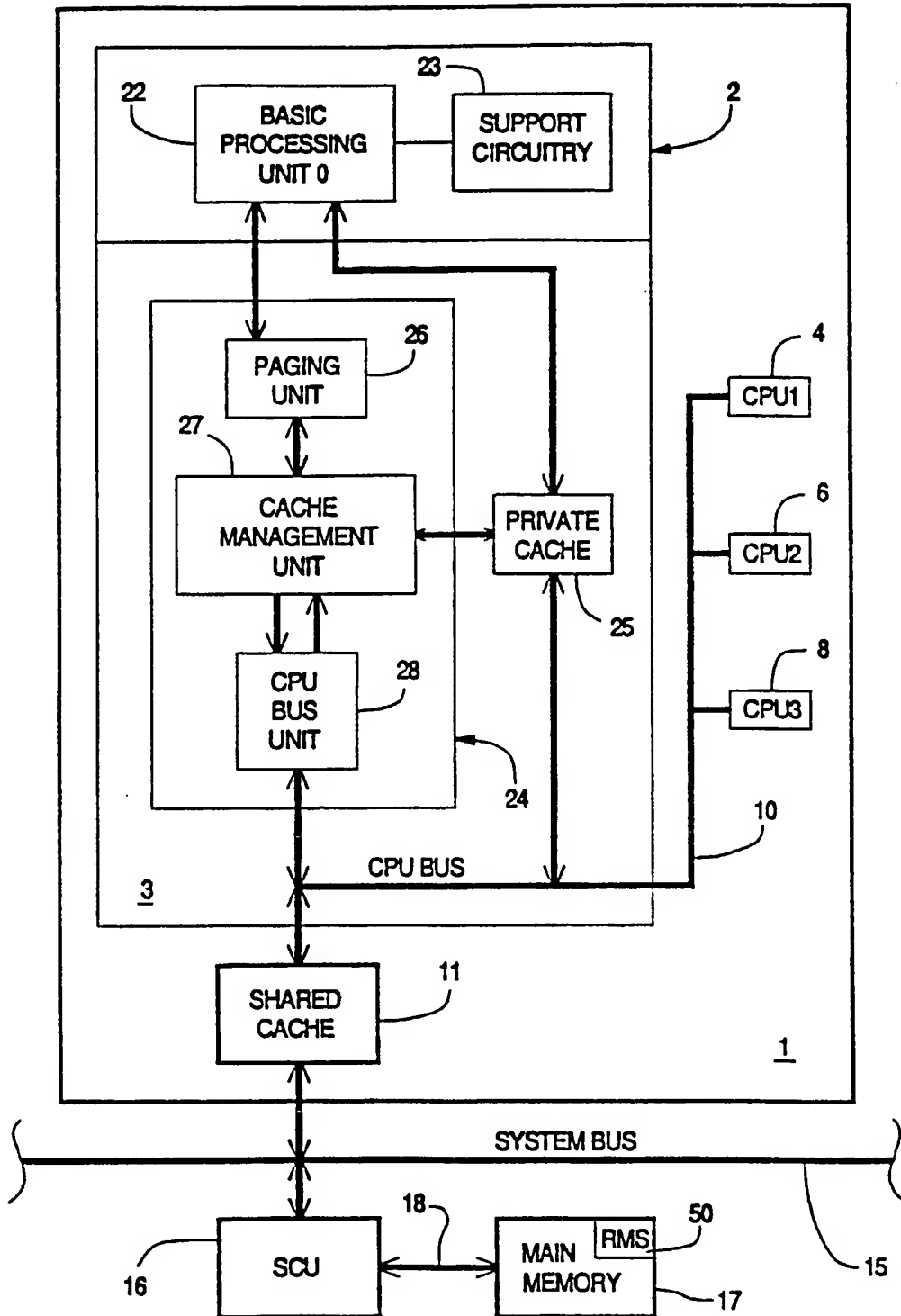
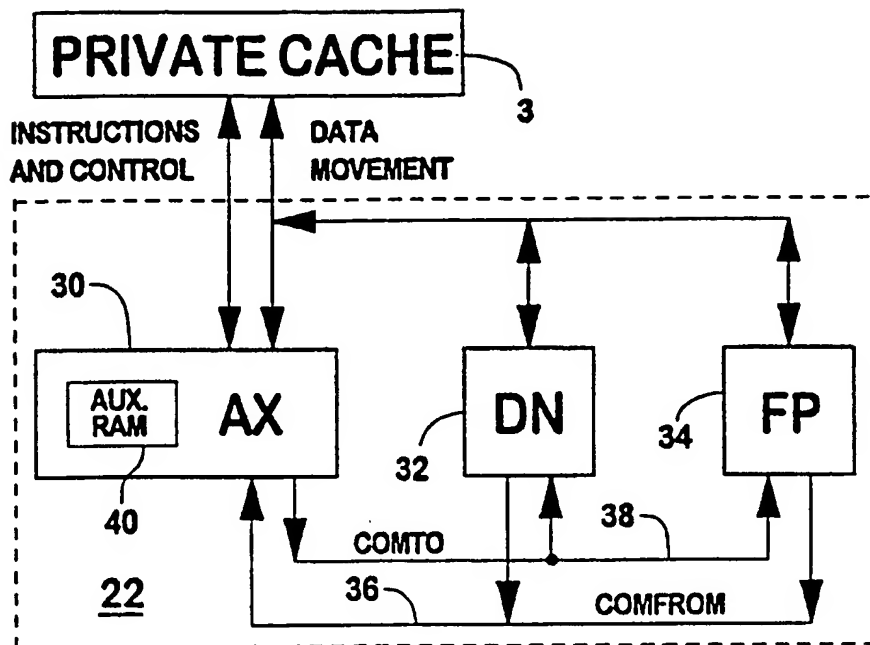
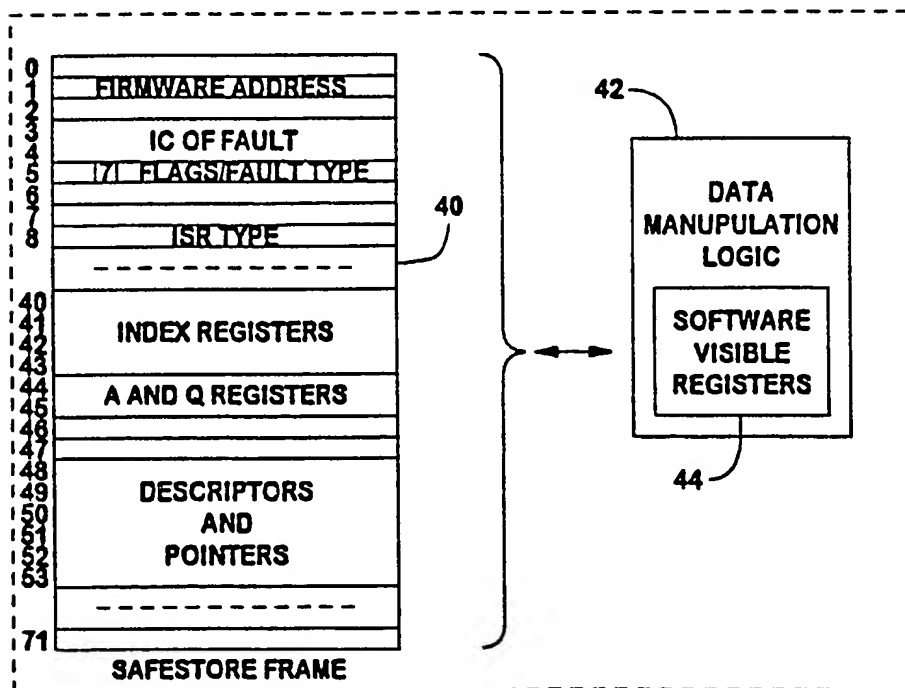
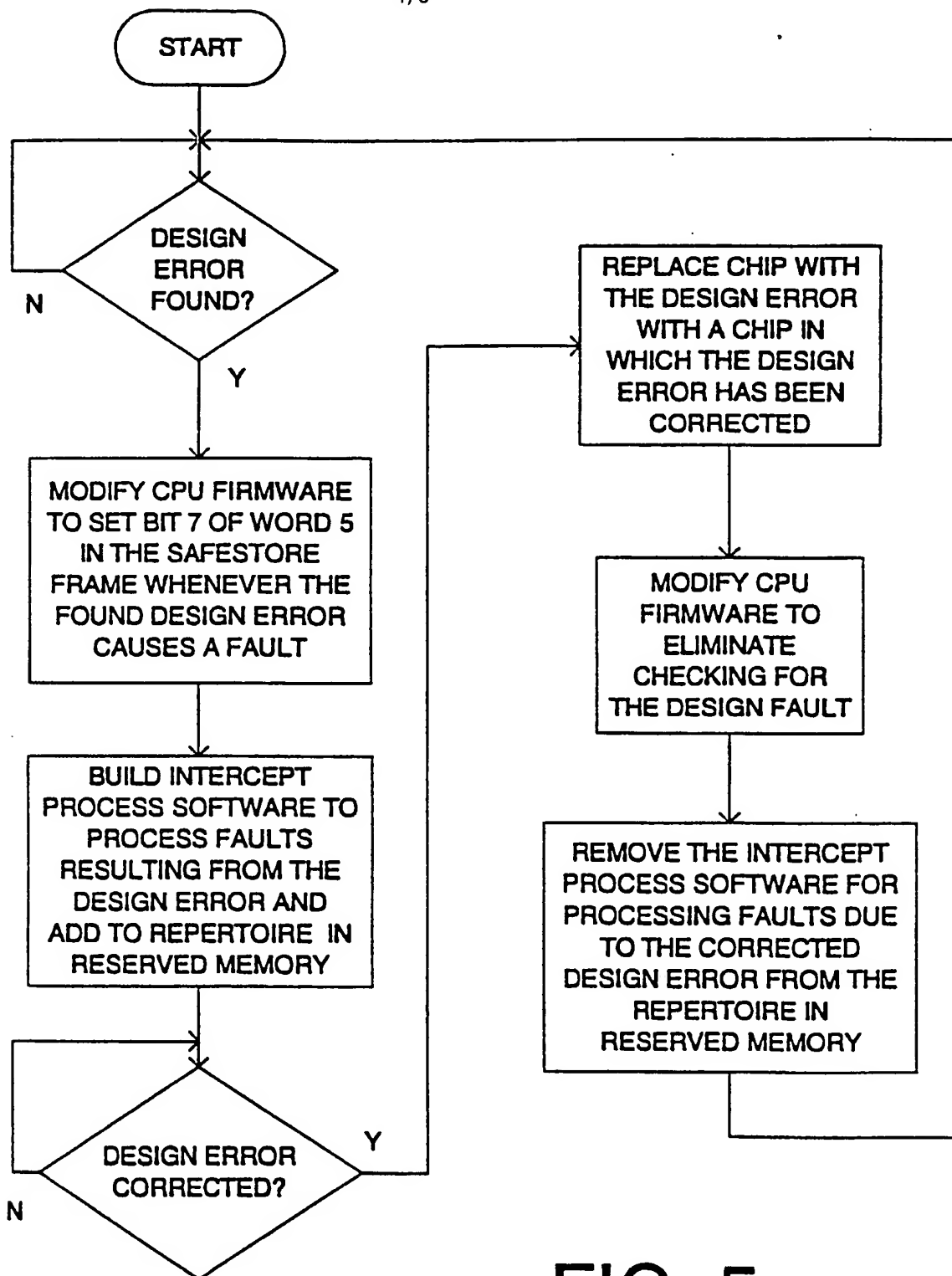


FIG. 2

**FIG. 3****FIG. 4**

**FIG. 5**

5/5

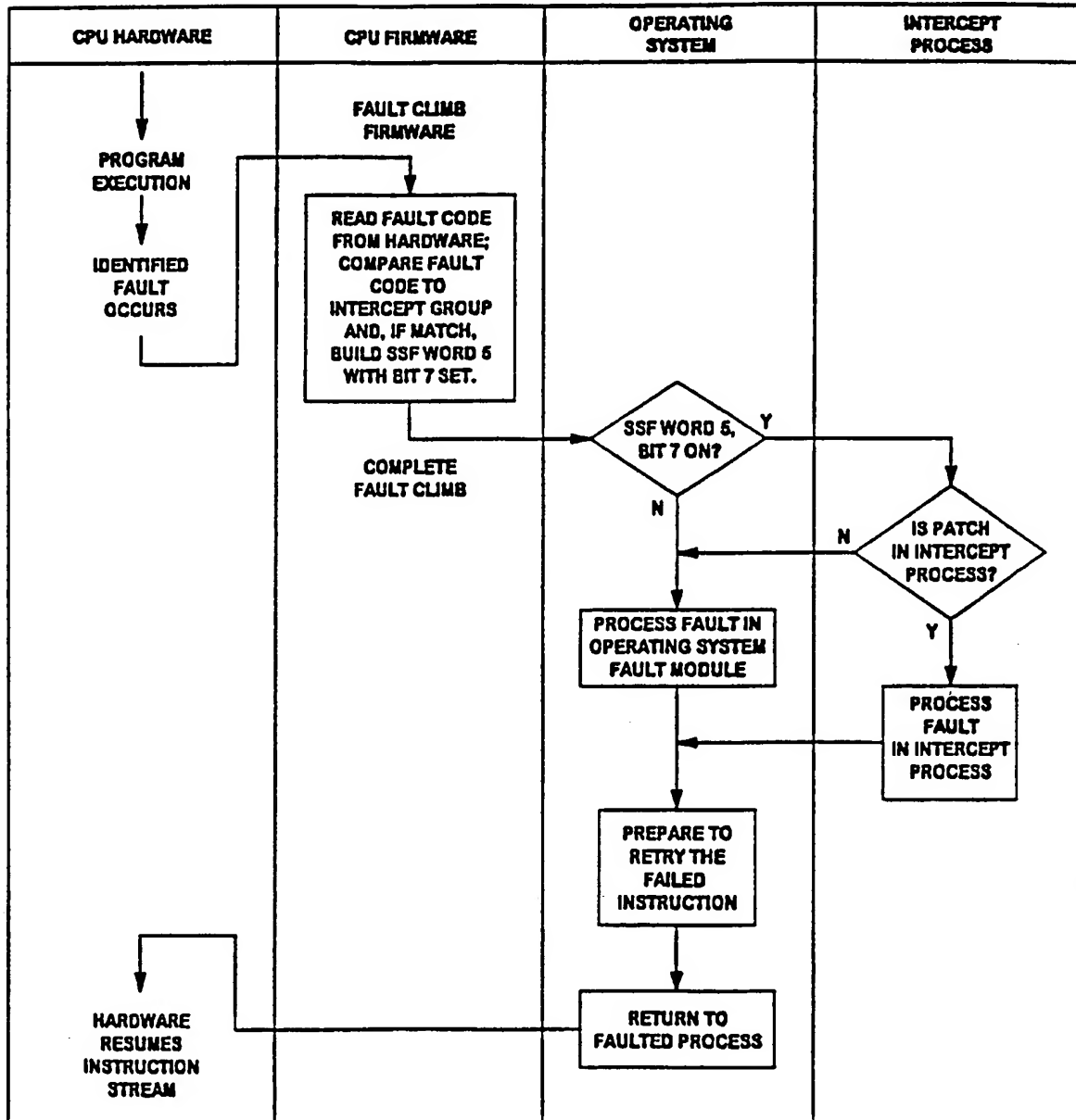


FIG. 6